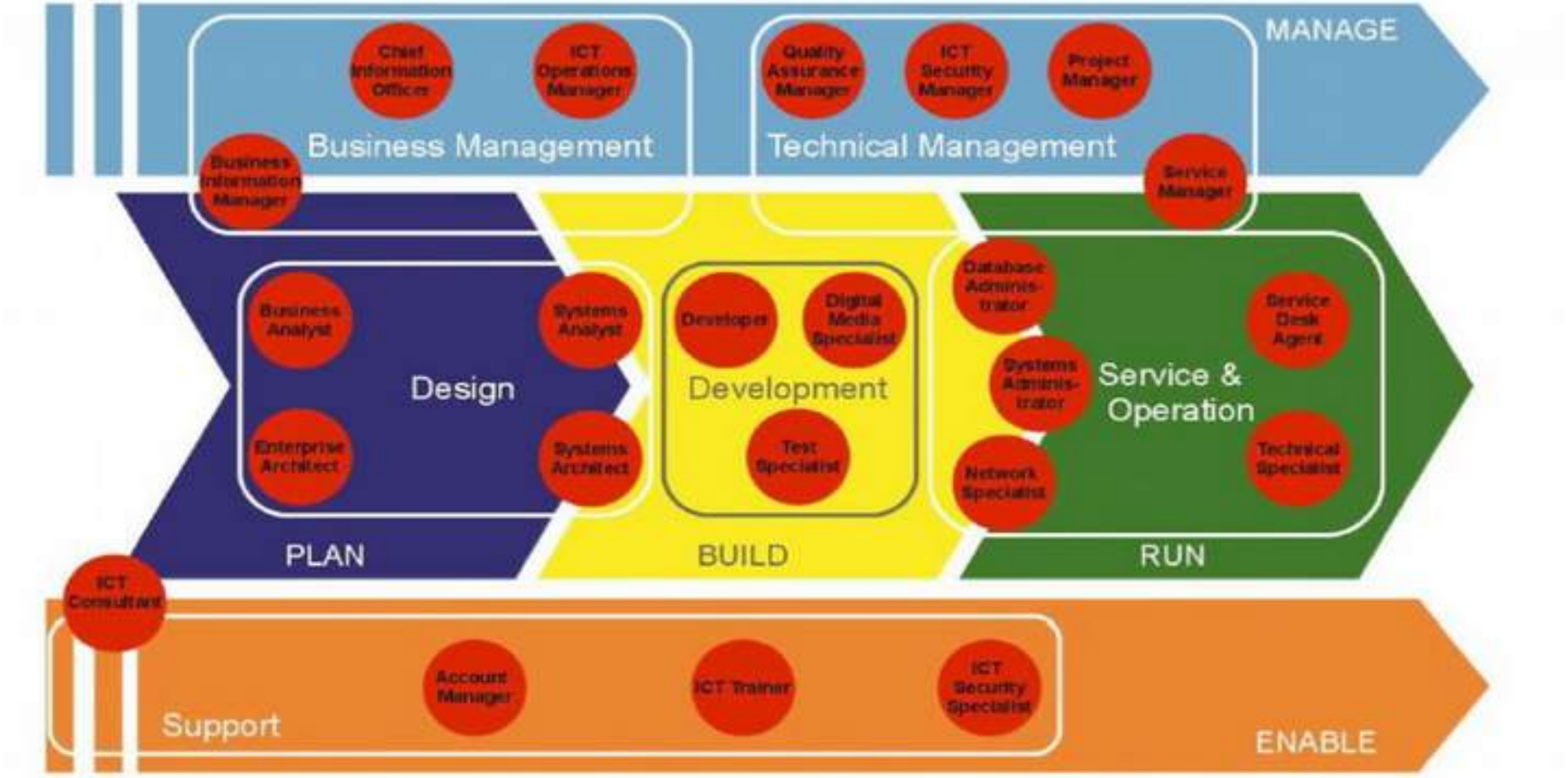
Tarkvara arhitektuur (ja ka andmebaasidest)

Gunnar Piho

Kasutan Robert Martin (Uncle Bob) ideid ja ka materjale

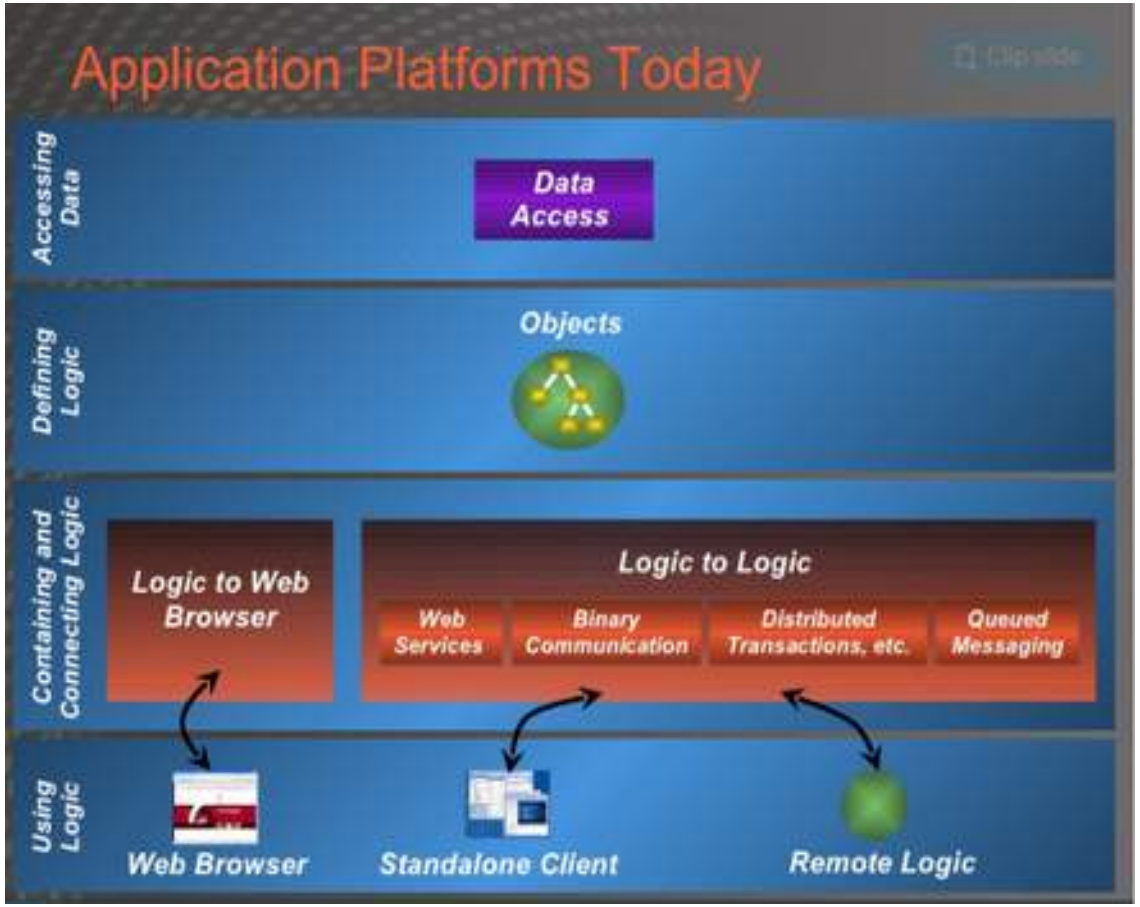
https://www.youtube.com/watch?v=0oGpWmS0aYQ

Millest räägin?

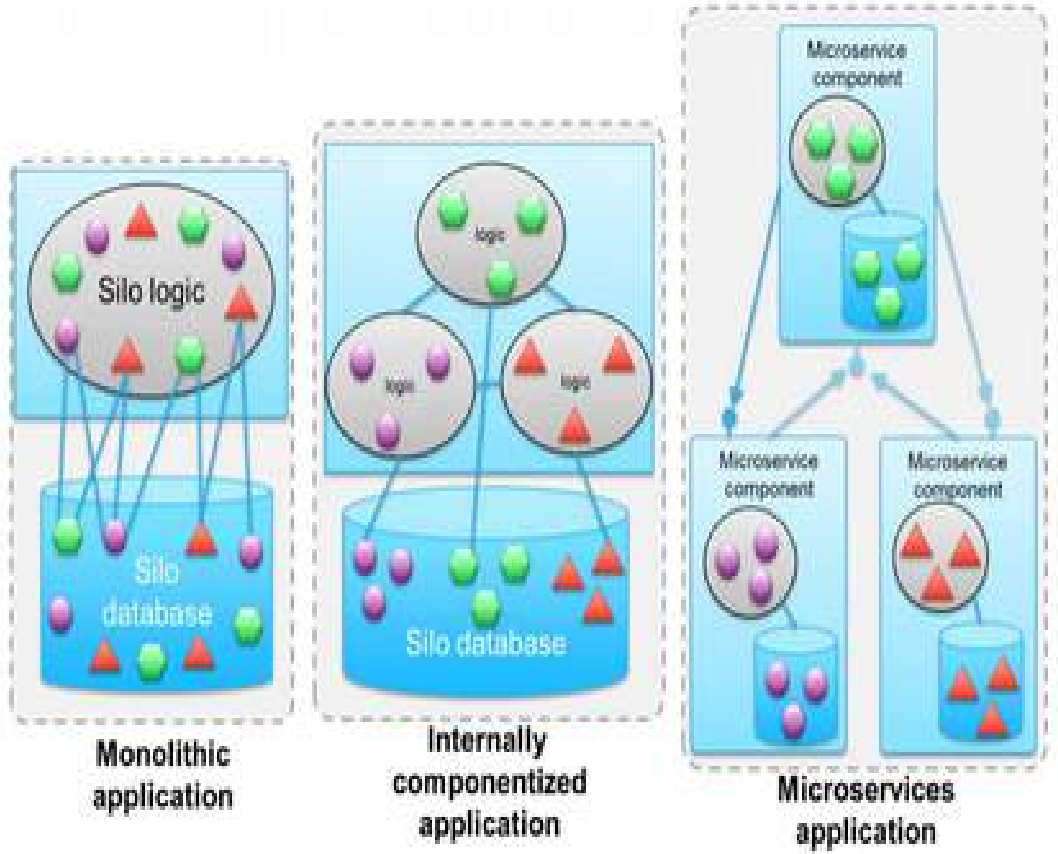


© http://www.ecompetences.eu/

Millest räägin?



© https://www.slideshare.net/kkorovkin/dev212-comparing-net-and-java-the-view-from-2006-presentation



©https://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

Kõigest räägin „code first“ võtmes

„Mission impossible“ kui teaduse arengu mootor

Sir Charles Antony Richard Hoare

Kõigepealt oli naine

Kõigepealt oli naine

1850

≈1850

Ada Lovelace

From Wikipedia, the free encyclopedia

Augusta Ada King-Noel, Countess of Lovelace (*née* **Byron**; 10 December 1815 – 27 November 1852) was an English mathematician and writer, chiefly known for her work on Charles Babbage's proposed mechanical general-purpose computer, the Analytical Engine. She was the first to recognise that the machine had applications beyond pure calculation, and published the first algorithm intended to be carried out by such a machine. As a result, she is often regarded as the first to recognise the full potential of a "computing machine" and the first computer programmer.[1][2][3]

Ja esimene arvuti, Z3 (Konrad Zuse) 1941

Ja siis tuli ...

1968

Software crisis

From Wikipedia, the free encyclopedia

Software crisis is a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could not be tackled. With the increase in the complexity of the software, many software problems arose because existing methods were insufficient.

The term "software crisis" was coined by some attendees at the first NATO Software Engineering Conference in 1968 at Garmisch, Germany.[1][2] Edsger Dijkstra's 1972 ACM Turing Award Lecture makes reference to this same problem:[3]

> The major cause of the software crisis is that the machines have become several orders of magnitude more powerfull To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.
>
> — Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM

The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

Ja sealt sai alguse ...
1975

The Mythical Man-Month

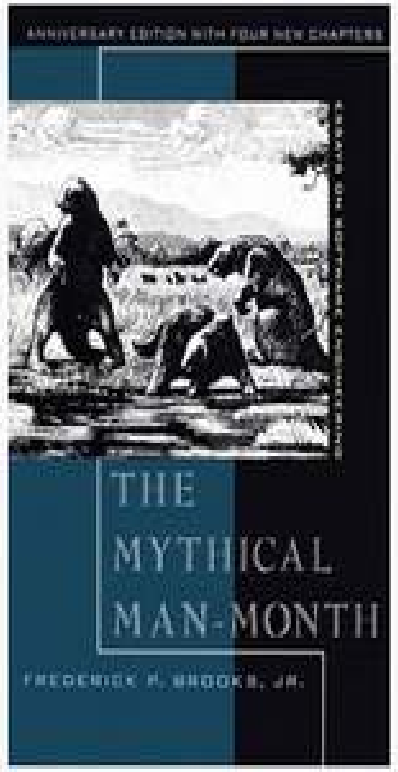
From Wikipedia, the free encyclopedia

The Mythical Man-Month: Essays on Software Engineering is a book on software engineering and project management by Fred Brooks, whose central theme is that "adding manpower to a late software project makes it later". This idea is known as Brooks's law, and is presented along with the second-system effect and advocacy of prototyping.

Brooks' observations are based on his experiences at IBM while managing the development of OS/360. He had added more programmers to a project falling behind schedule, a decision that he would later conclude had, counter-intuitively, delayed the project even further. He also made the mistake of asserting that one project—writing an ALGOL compiler—would require six months, regardless of the number of workers involved (it required longer). The tendency for managers to repeat such errors in project development led Brooks to quip that his book is called "The Bible of Software Engineering", because "everybody quotes it, some people read it, and a few people go by it".[1] The book is widely regarded as a classic on the human elements of software engineering.[2]

The work was first published in 1975 (ISBN 0-201-00650-2), reprinted with corrections in 1982, and republished in an anniversary edition with four extra chapters in 1995 (ISBN 0-201-83595-9), including a reprint of the essay "No Silver Bullet" with commentary by the author.

The Mythical Man-Month



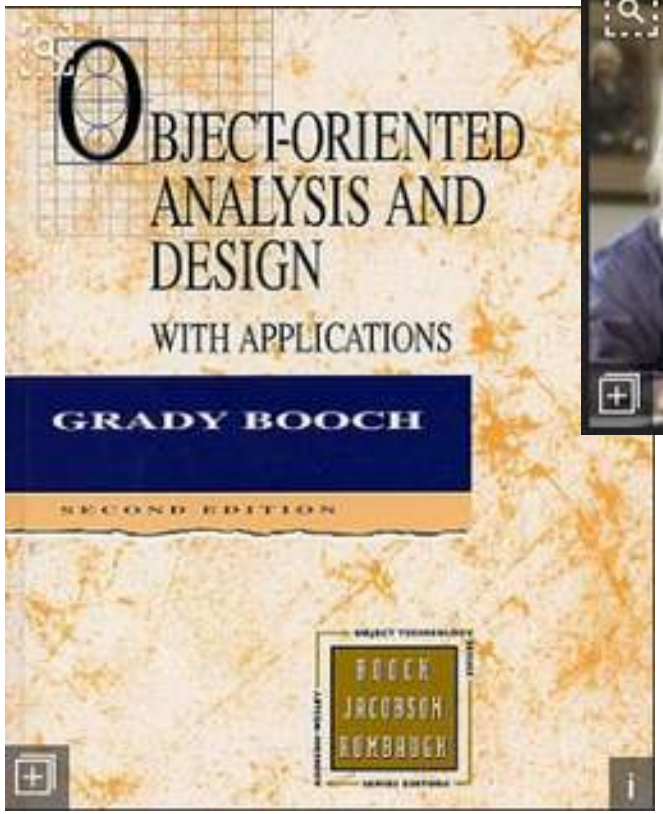
Fred Brooks

2007 photo

Objektorienteeritud ... 1980

Objektorienteeritud ...

≈ 1980



Object-oriented analysis [edit]

The purpose of any analysis activity in the software life-cycle is to create a model of the system's functional requirements that is independent of implementation constraints.

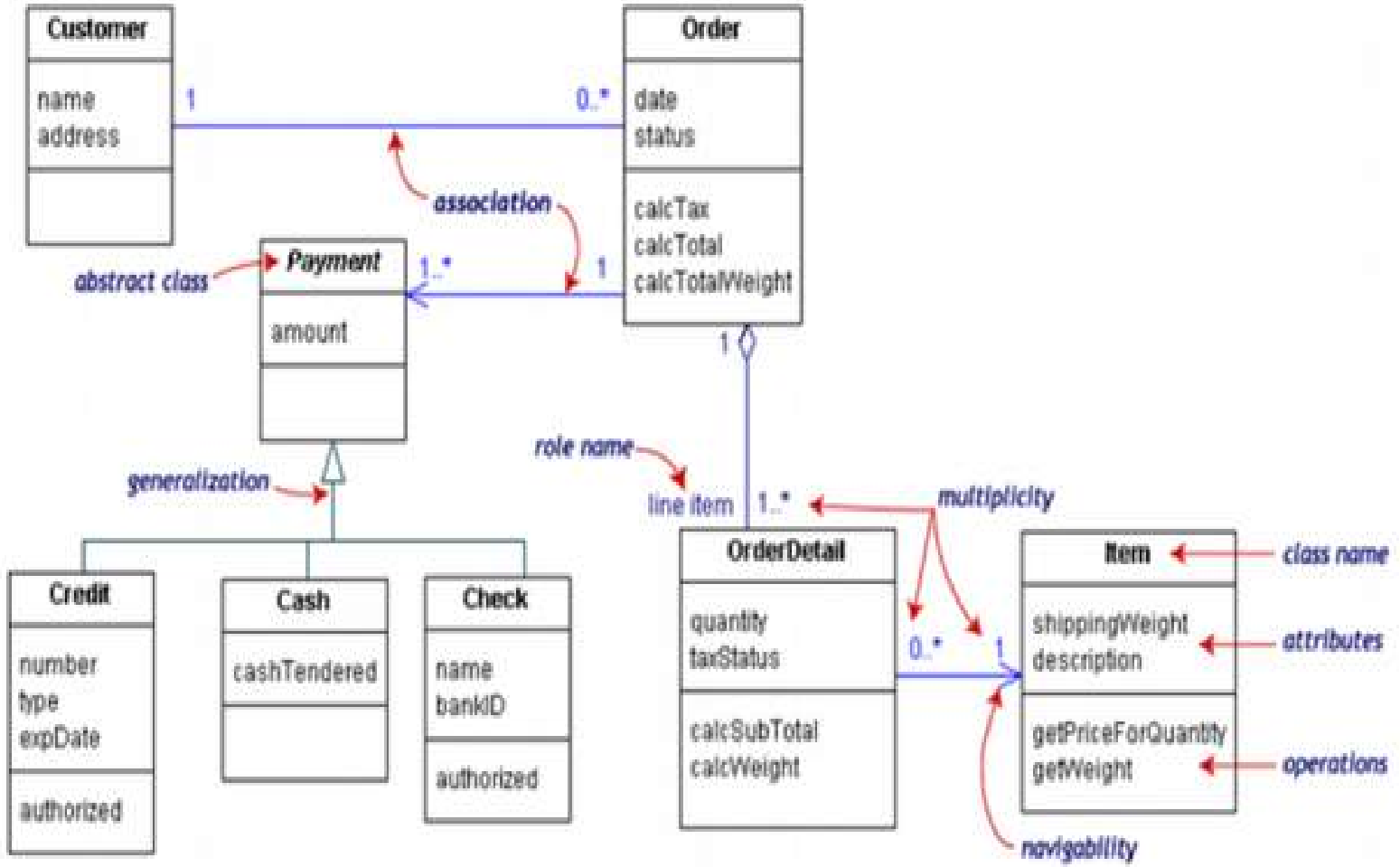
The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors (processes) and states (data) modeled after real world objects that the system interacts with. In other or traditional analysis methodologies, the two aspects: processes and data are considered separately. For example, data may be modeled by ER diagrams, and behaviors by flow charts or structure charts.

The primary tasks in object-oriented analysis (OOA) are:

- Find the objects
- Organize the objects
- Describe how the objects interact
- Define the behavior of the objects
- Define the internals of the objects

Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish. Object models describe the names, class relations (e.g. Circle is a subclass of Shape), operations, and properties of the main objects. User-interface mockups or prototypes can also be created to help understanding.[5]

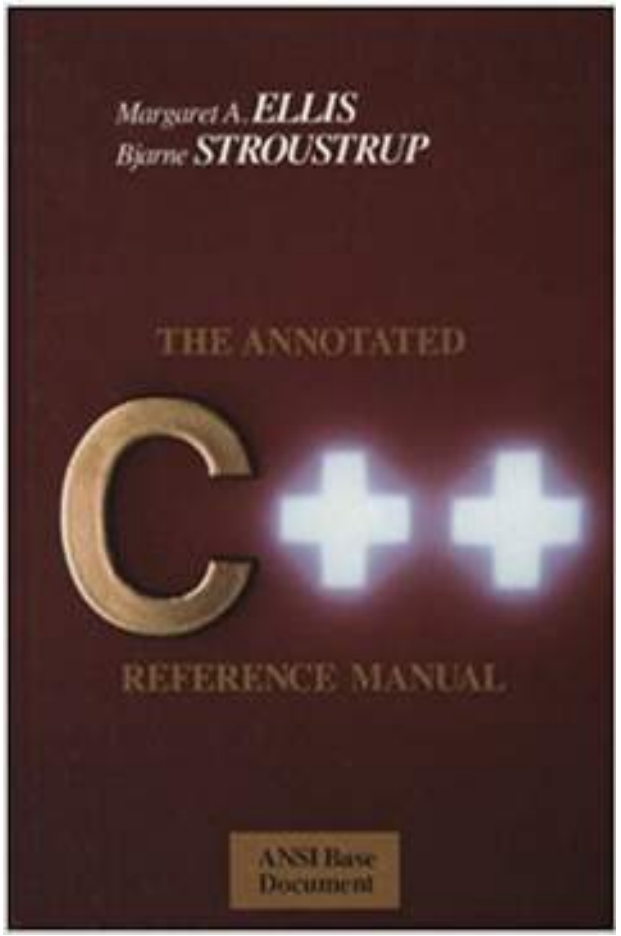
Objektid



© http://edn.embarcadero.com/article/31863

Ja siis algas Java ajastu …

1990



Bjarne Stroustrup

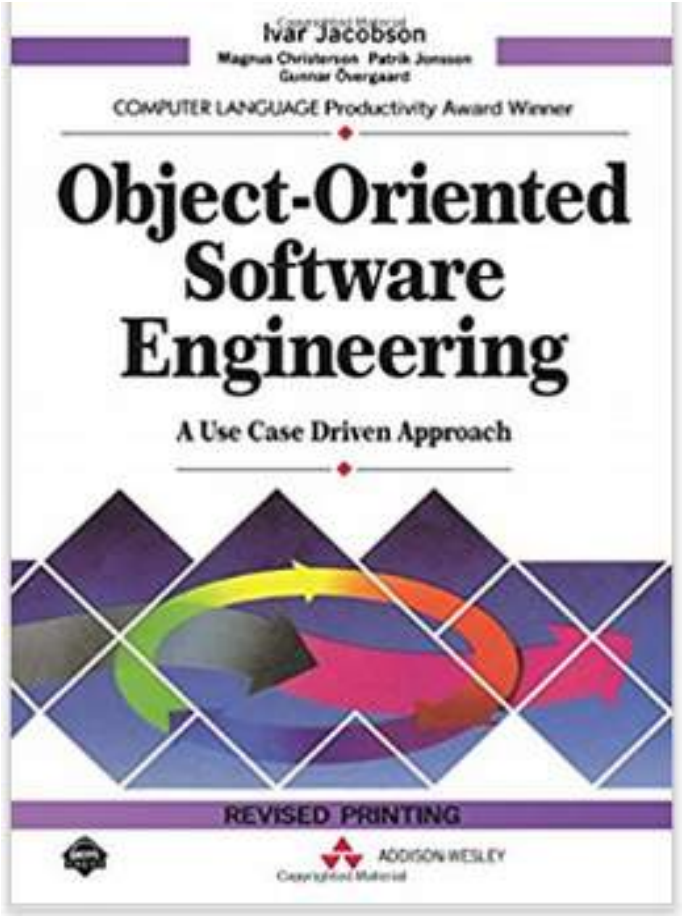
Margaret A. Ellis

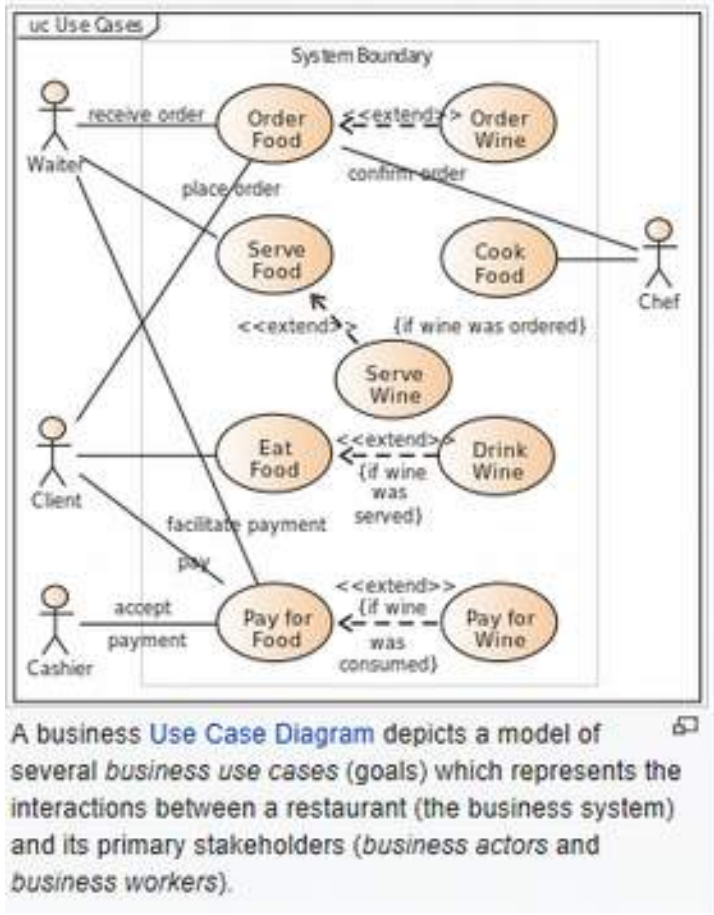
Ja siis algas Java ajastu …

≈ 1990

Use Case maania ... 1992

Ivar Jakobson





A business Use Case Diagram depicts a model of several *business use cases* (goals) which represents the interactions between a restaurant (the business system) and its primary stakeholders (*business actors* and *business workers*).

Mis oli Use Case juures oli väärtuslik ?

Cockburn describes a more detailed structure for a use case, but permits it to be simplified when less detail is needed. His fully dressed use case template lists the following fields:[9]

- Title: "an active-verb goal phrase that names the goal of the primary actor"[10]
- Primary Actor
- Goal in Context
- Scope
- Level
- Stakeholders and Interests
- Precondition
- Minimal Guarantees
- Success Guarantees
- Trigger
- Main Success Scenario
- Extensions
- Technology & Data Variations List

Martin Fowler states "There is no standard way to write the content of a use case, and different formats work well in different cases."[8]:100 He describes "a common style to use" as follows:[8]:101

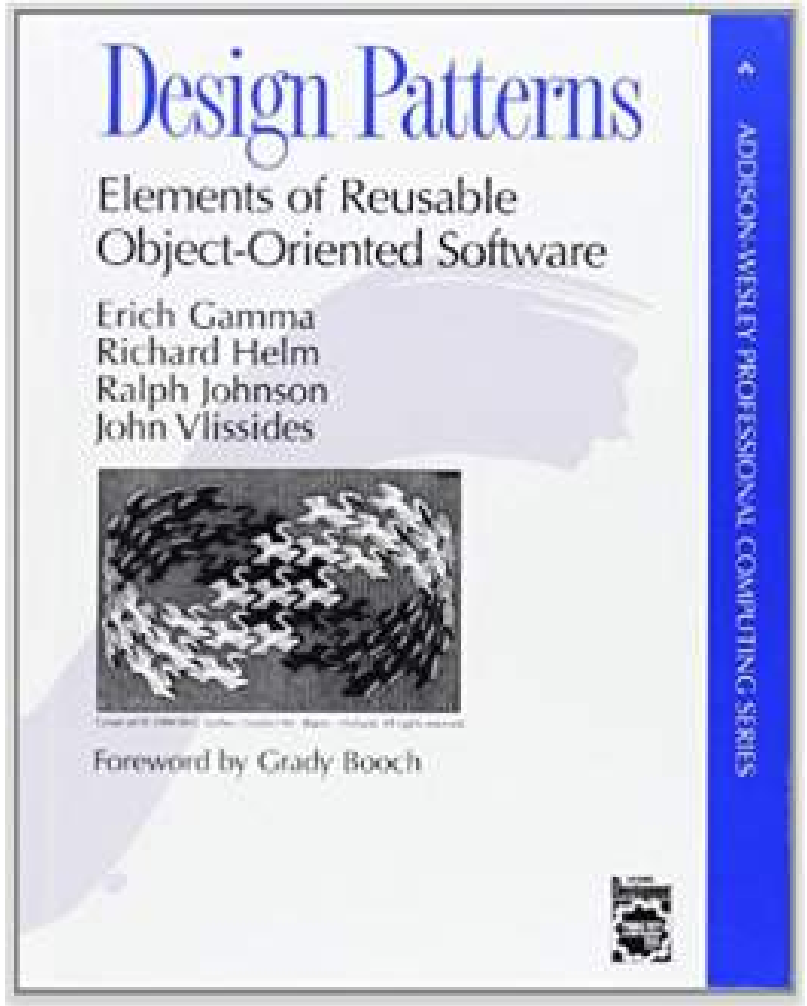
- Title: "goal the use case is trying to satisfy"[8]:101
- Main Success Scenario: numbered list of steps[8]:101
  - Step: "a simple statement of the interaction between the actor and a system"[8]:101
- Extensions: separately numbered lists, one per Extension[8]:101
  - Extension: "a condition that results in different interactions from .. the main success scenario". An extension from main step 3 is numbered 3a, etc.[8]:101

The Fowler style can also be viewed as a simplified variant of the Cockburn template.

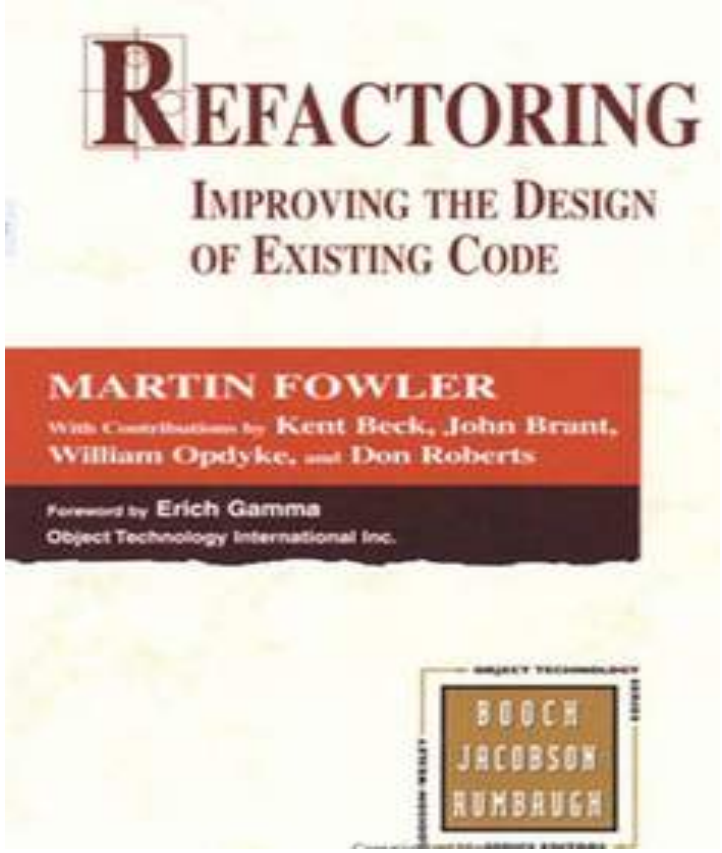
© Vikipedia

© Uncle Bob

Go4 -Design Patterns 1994



Ja tagasi koodi (juurte) juurde ...
1999



© https://martinfowler.com/

Ja minimaalne arendusprotsess ... 2000

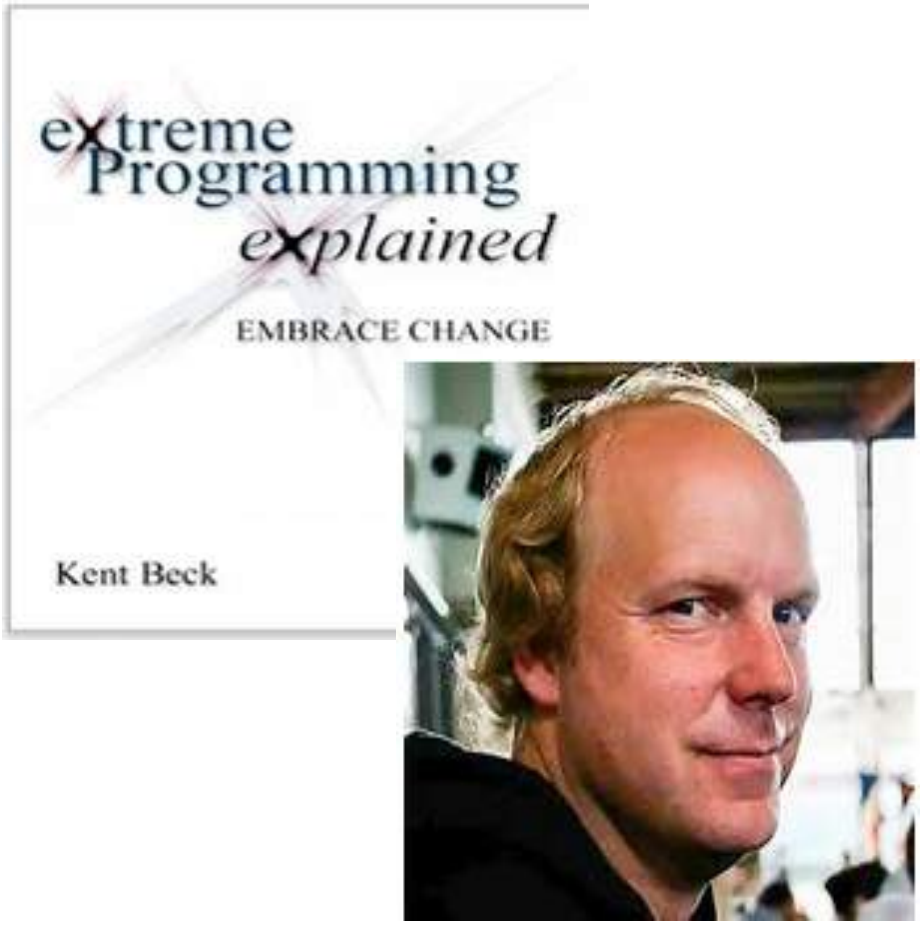
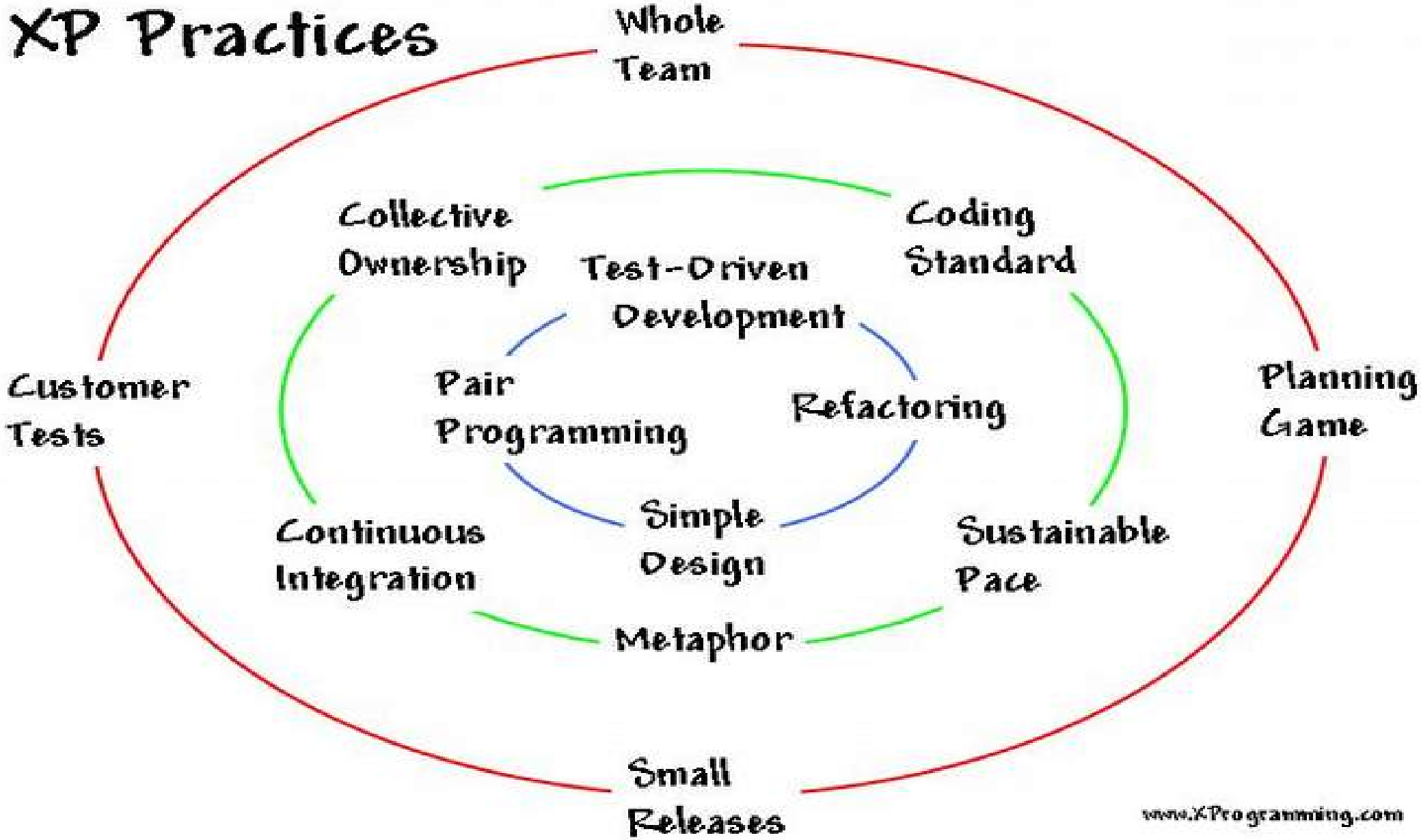


Table 2. The "Extreme" in Extreme Programming.

Common Sense Practice	XP Extreme	XP Implementation
Code reviews	Review code all the time	Pair programming
Testing	Test all the time, even by the customers	Unit testing, Functional testing
Design	Make design part of everybody's daily business	Refactoring
Simplicity	Always leave the system with the simplest design that supports its current functionality	The simplest thing that could possibly work
Architecture	Everybody will work to refine the architecture all the time	Metaphor
Integration testing	Integrate and test several times a day	Continuous integration
Short iterations	Make iterations really, really short -- seconds and minutes and hours, not weeks and months and years	Planning game

© http://csis.pace.edu/~marchese/CS616/Agile/XP/xp-cmm-paper.pdf


XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Customer Tests
Pair Programming
Refactoring
Planning Game
Continuous Integration
Simple Design
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com

Agile Manifesto 2001

Why Software Fails?

2005

We waste billions of dollars each year on entirely

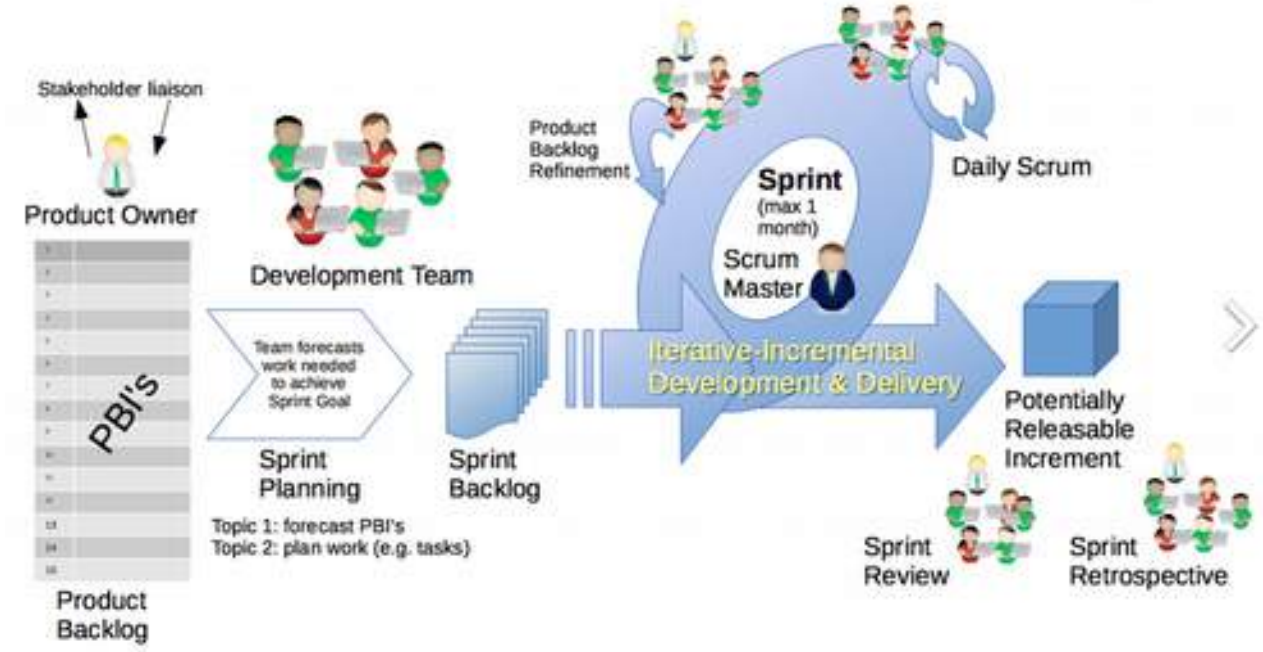
Why do projects fail so often?

Among the most common factors:

- Unrealistic or unarticulated project goals
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project's status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Inability to handle the project's complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

- Projects running over-budget
- Projects running over-time
- Software was very inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

Siis vallutas meid Scrum ... 2002



Jumal olgu meile armuline

© https://en.wikipedia.org/wiki/Scrum_%28software_development%29#/media/File:Scrum_Framework.png

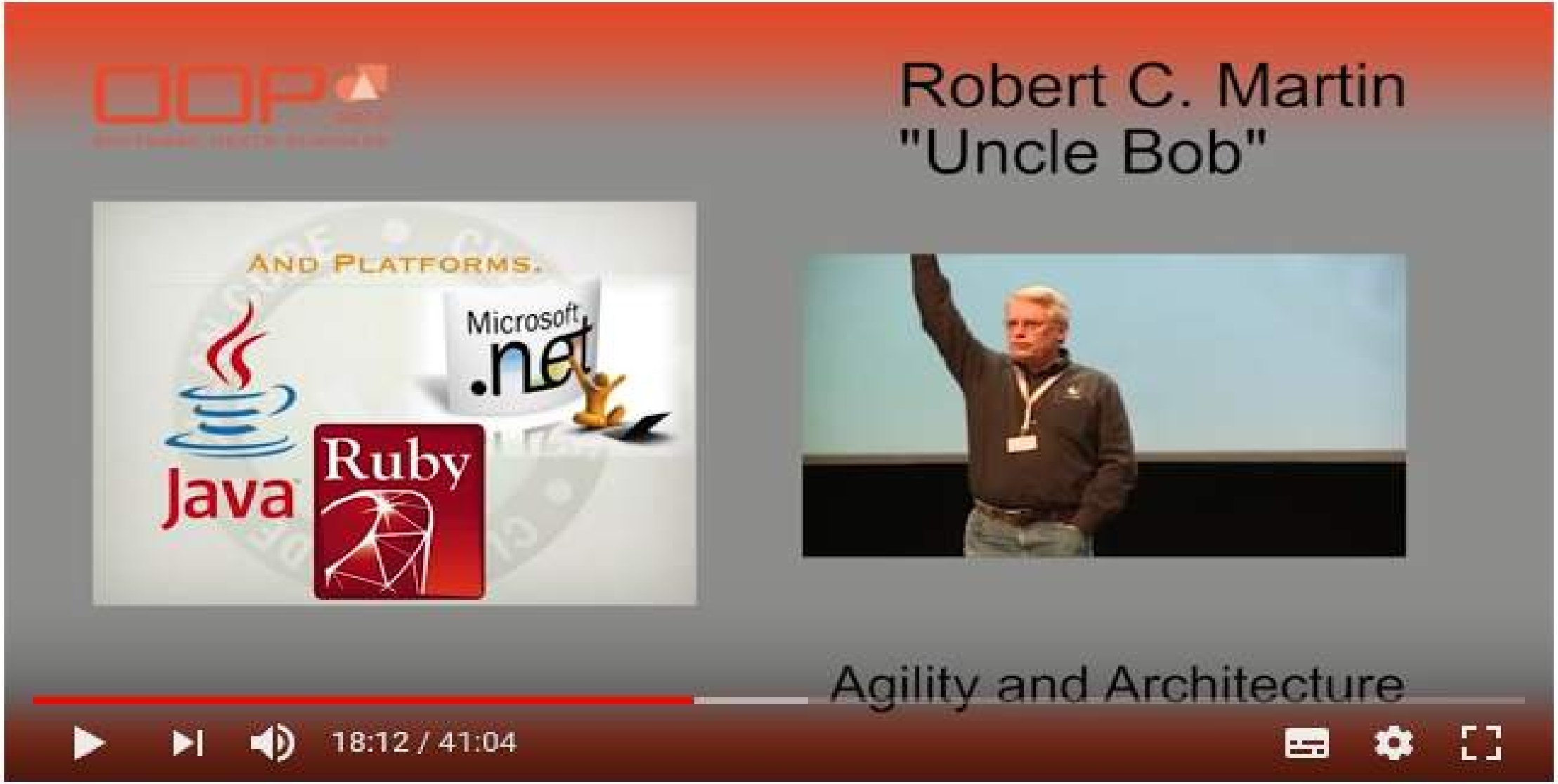
Siis hakkasimme kuulutama tõde

YANGI – You aren’t gonna nee

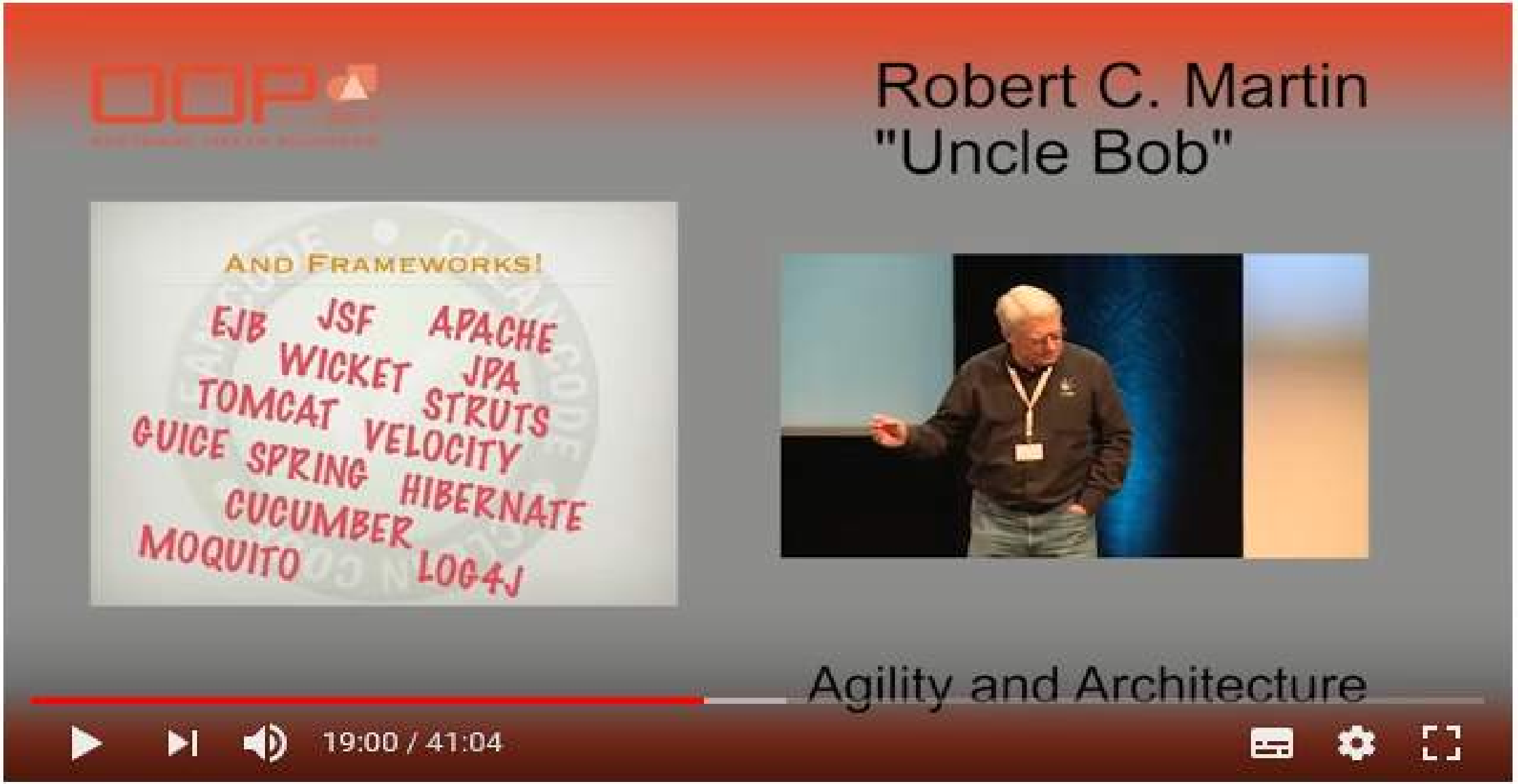
KISS – Keep it simple and stu

No BUFD – Big Design Up Fro

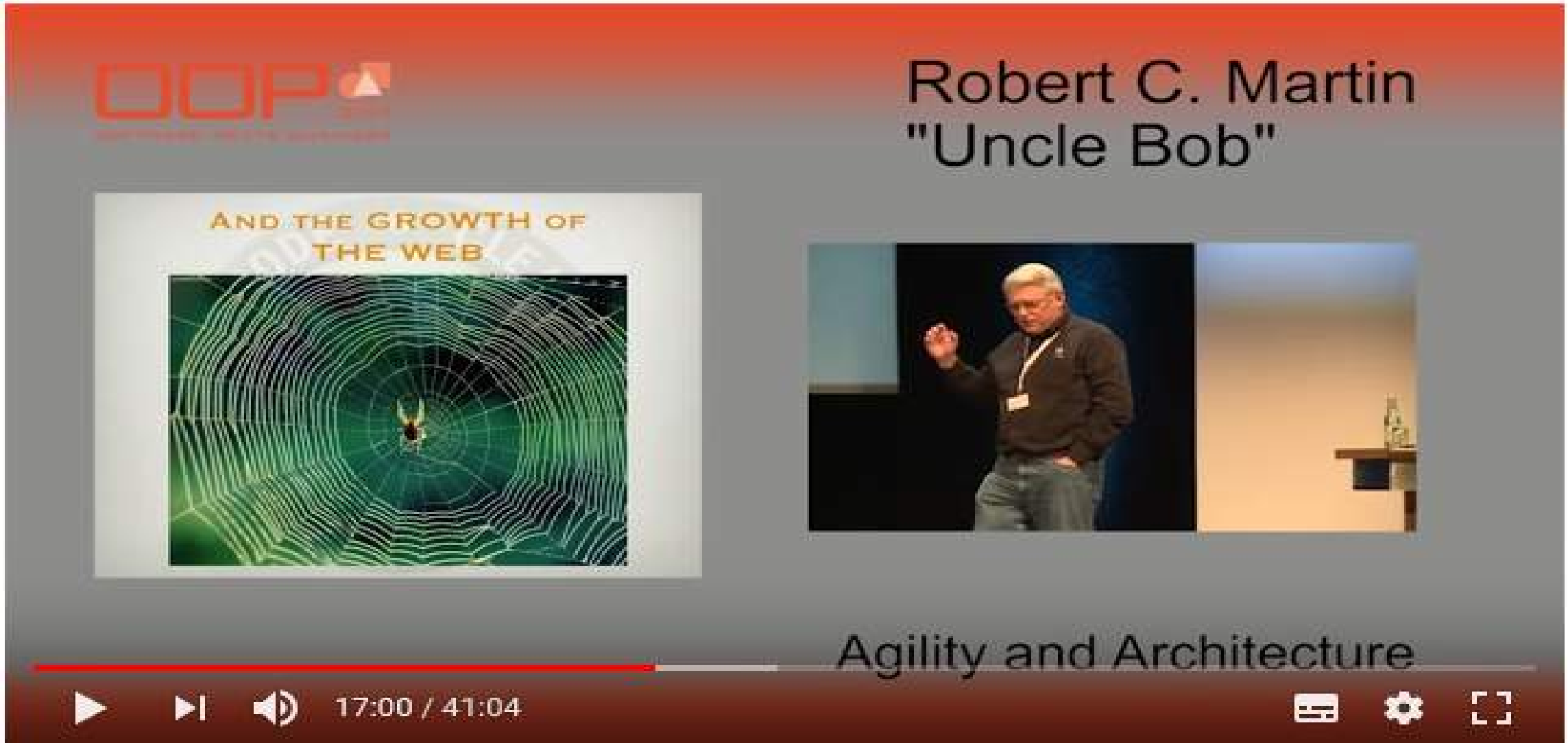
Ja arendusplatvormid



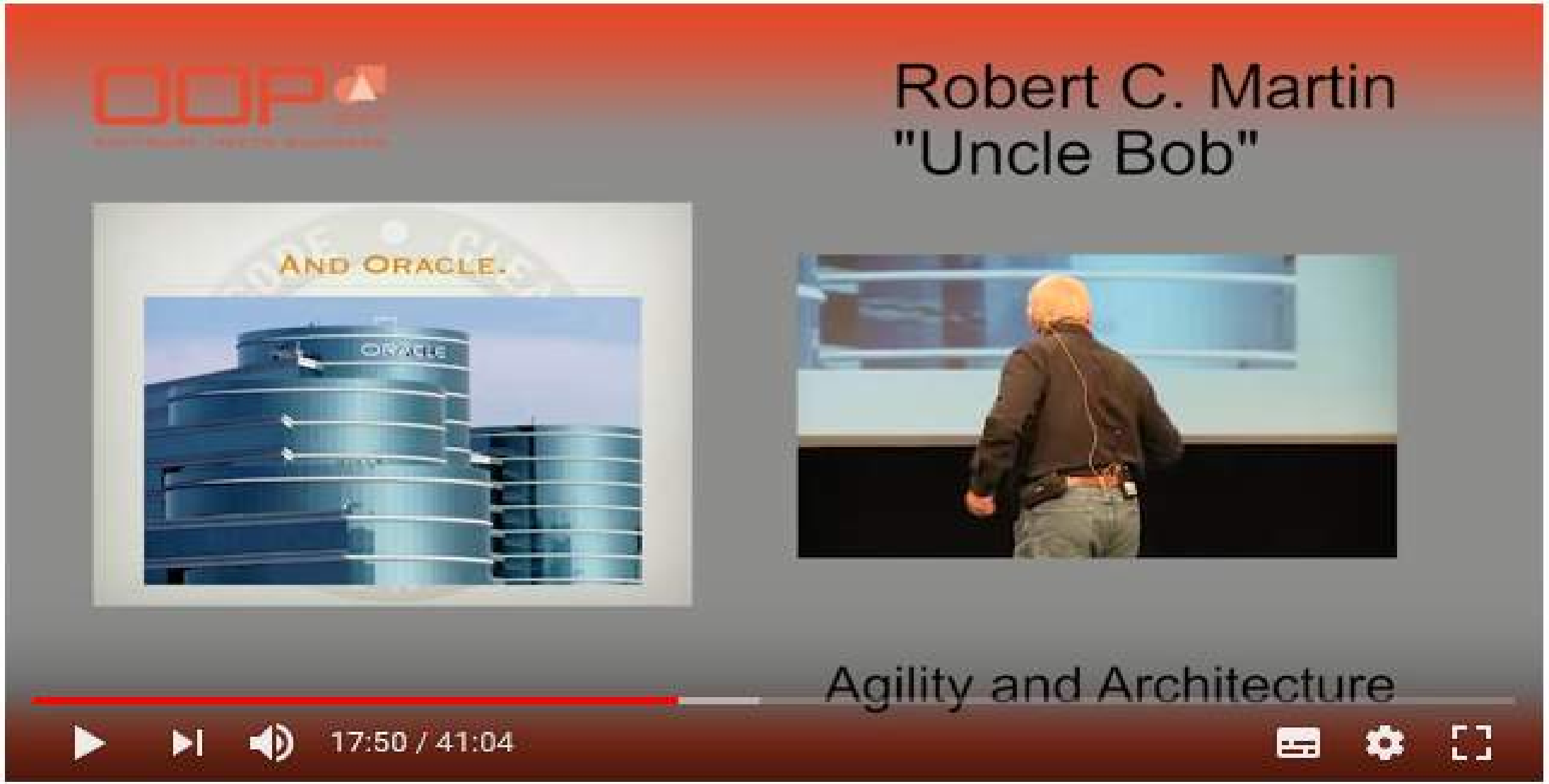
Ja raamistikud (ehk siis järjekordne BAD)



Vahepeal oli meid vallutanud võrk

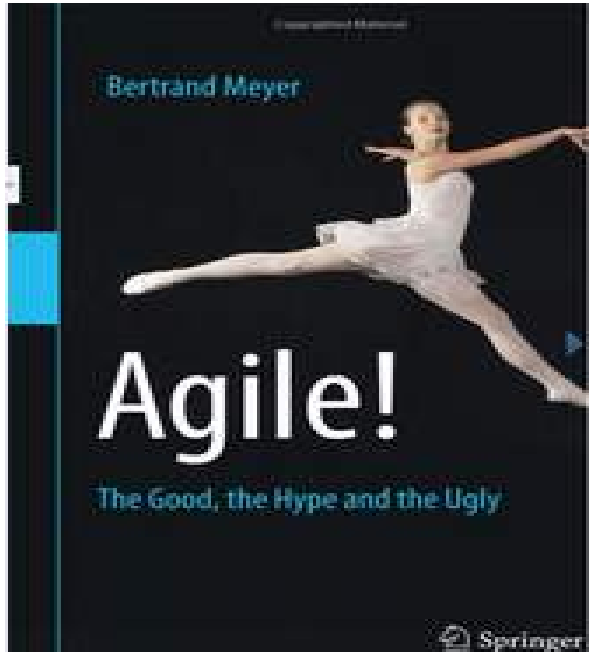


Ja ORACLE kuulutas, et andmebaas on interneti arengu mootor



.COM mulli lõhkemine

Ja me oleme jälle tagasi aastas 1968... 2014
Agile! The Good, the Hype and the Ugly

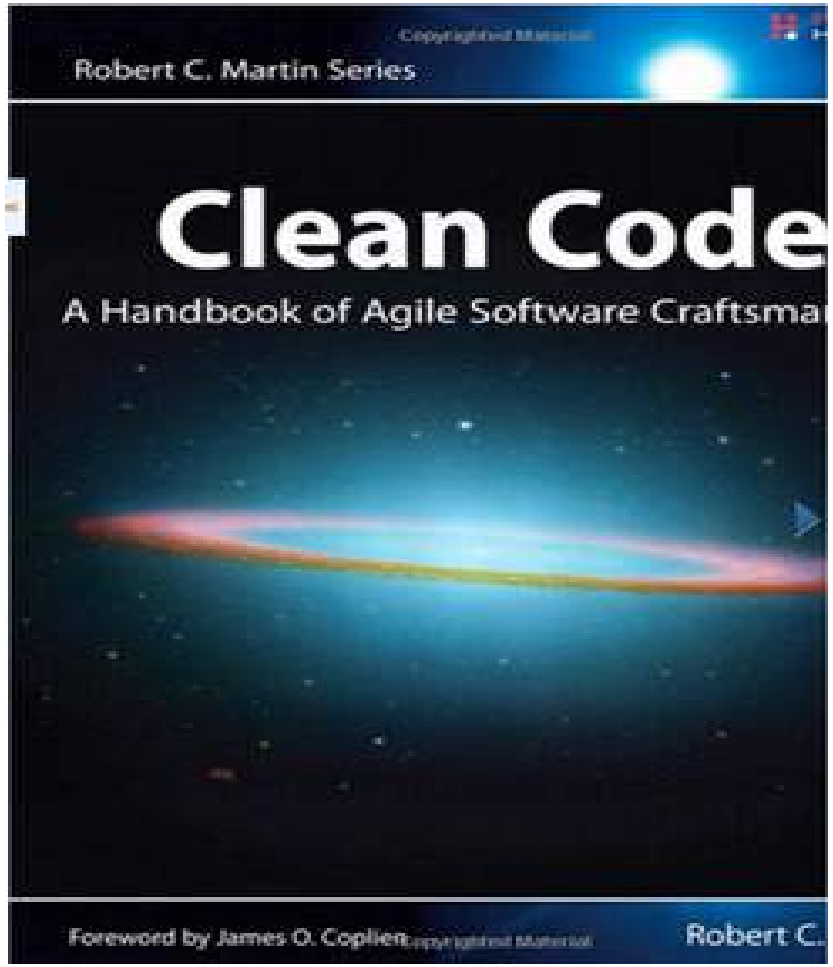


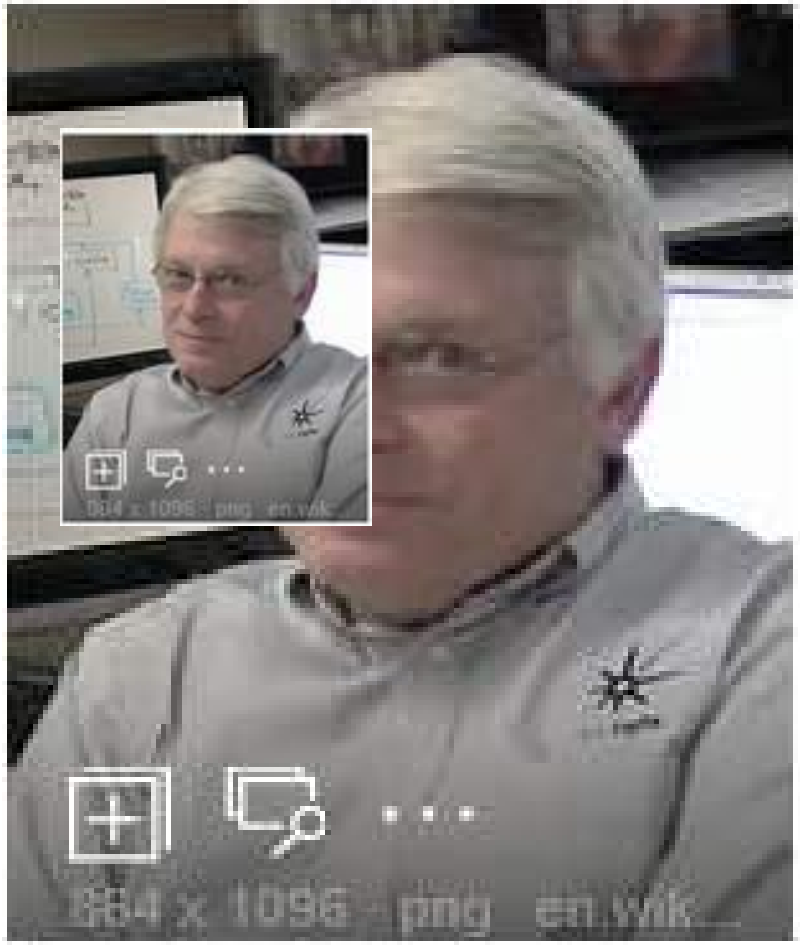
Fashion, Politics, or Engineering?

Software methodology is a strange field. In principle it should be a science-based engineering discipline, but in practice it looks at times like the fashion industry, and at time like politics.

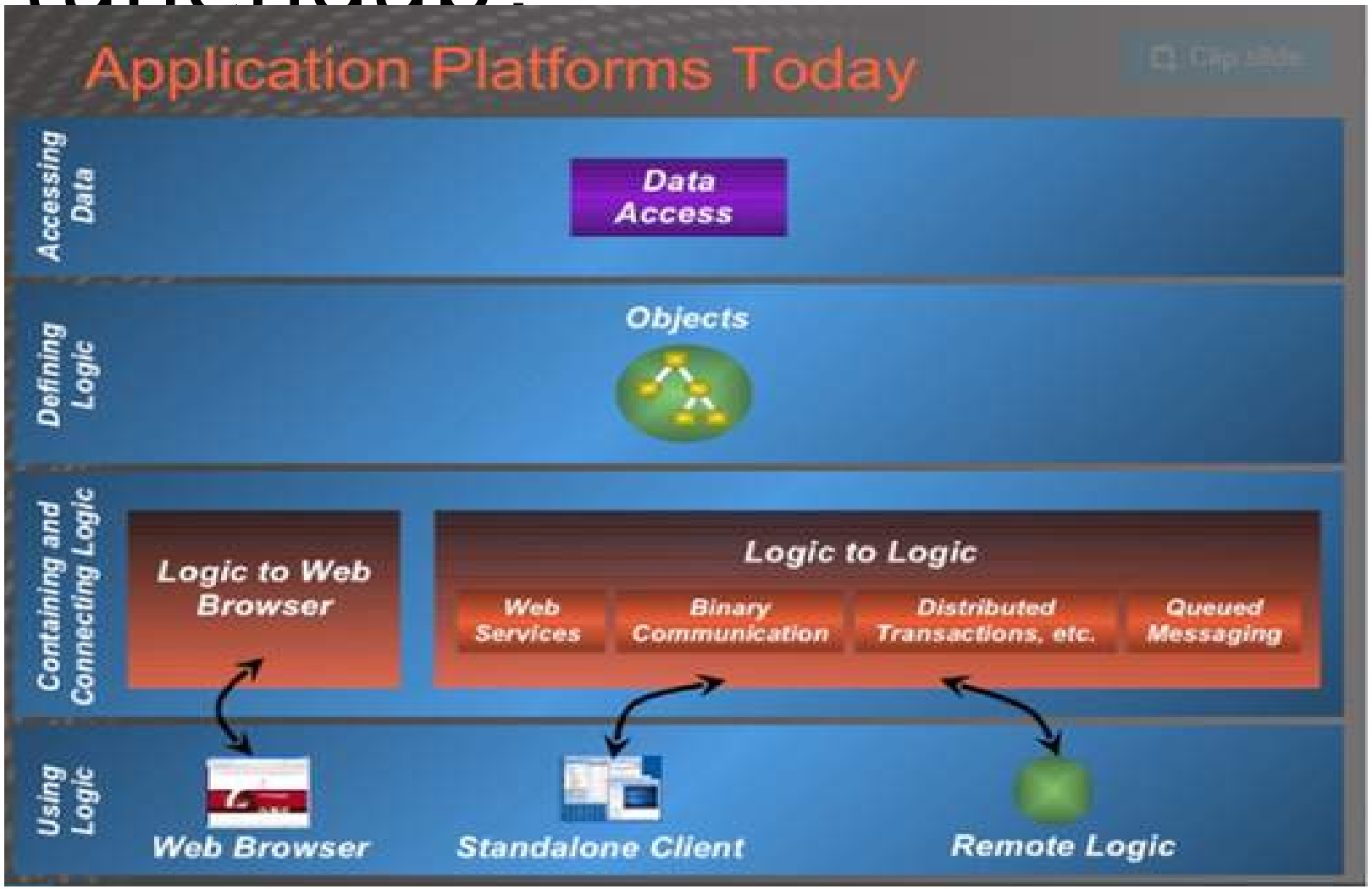
© http://www.drdobbs.com/architecture-and-design/methods-need-theory/219100242

Clean Code First (= testitav ja XXXignorant)

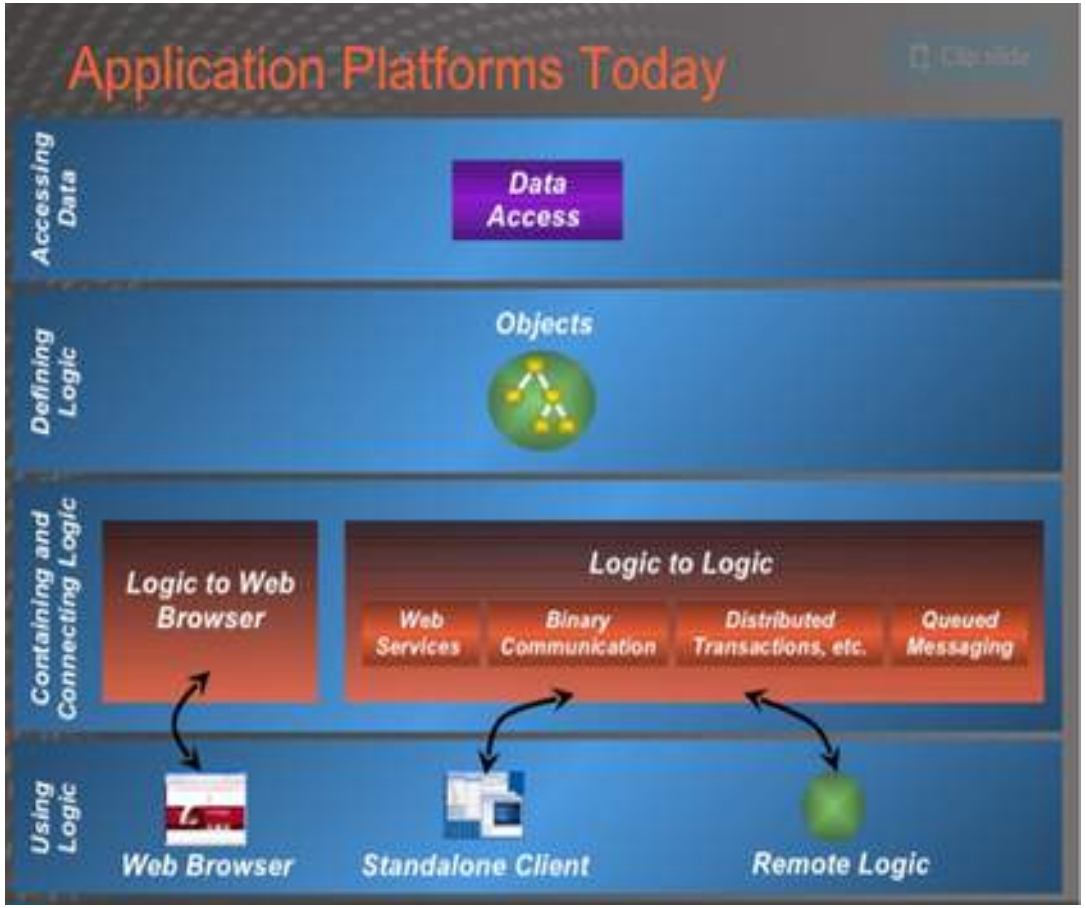




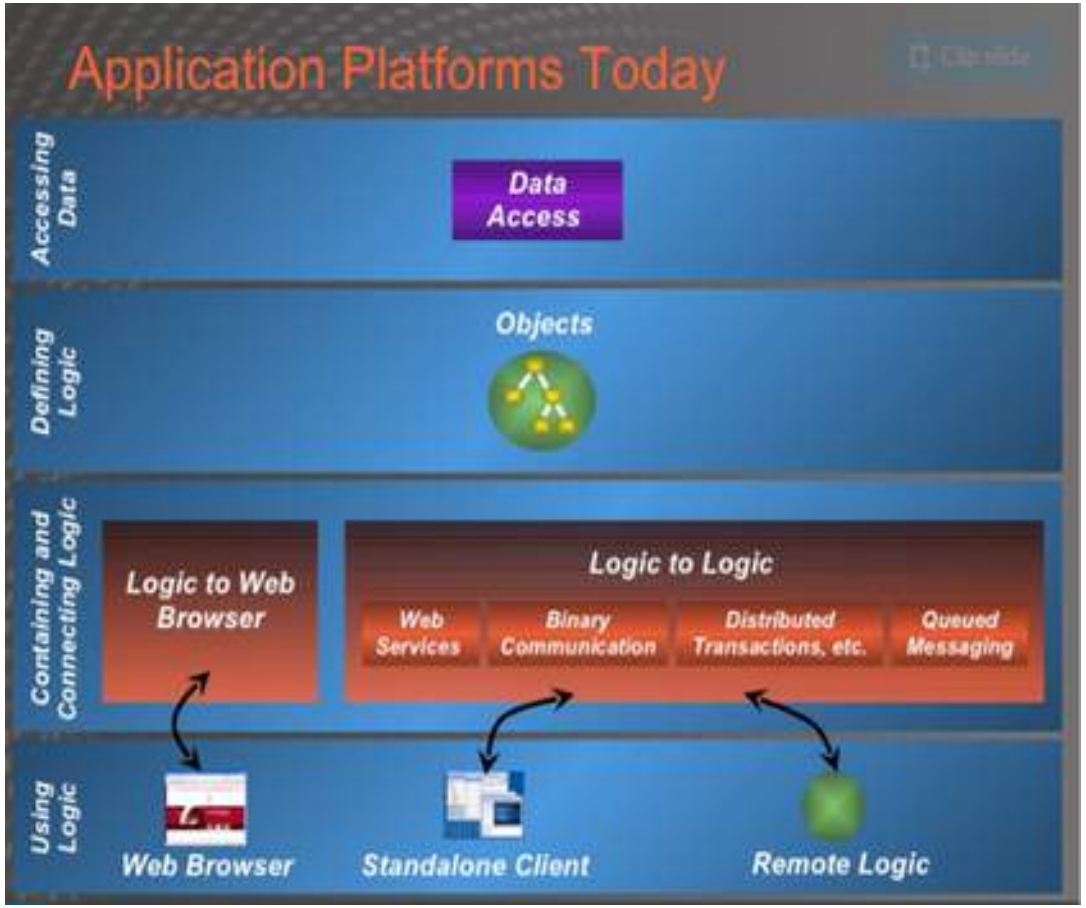
Mida see „Clean Code First“ tähendab?



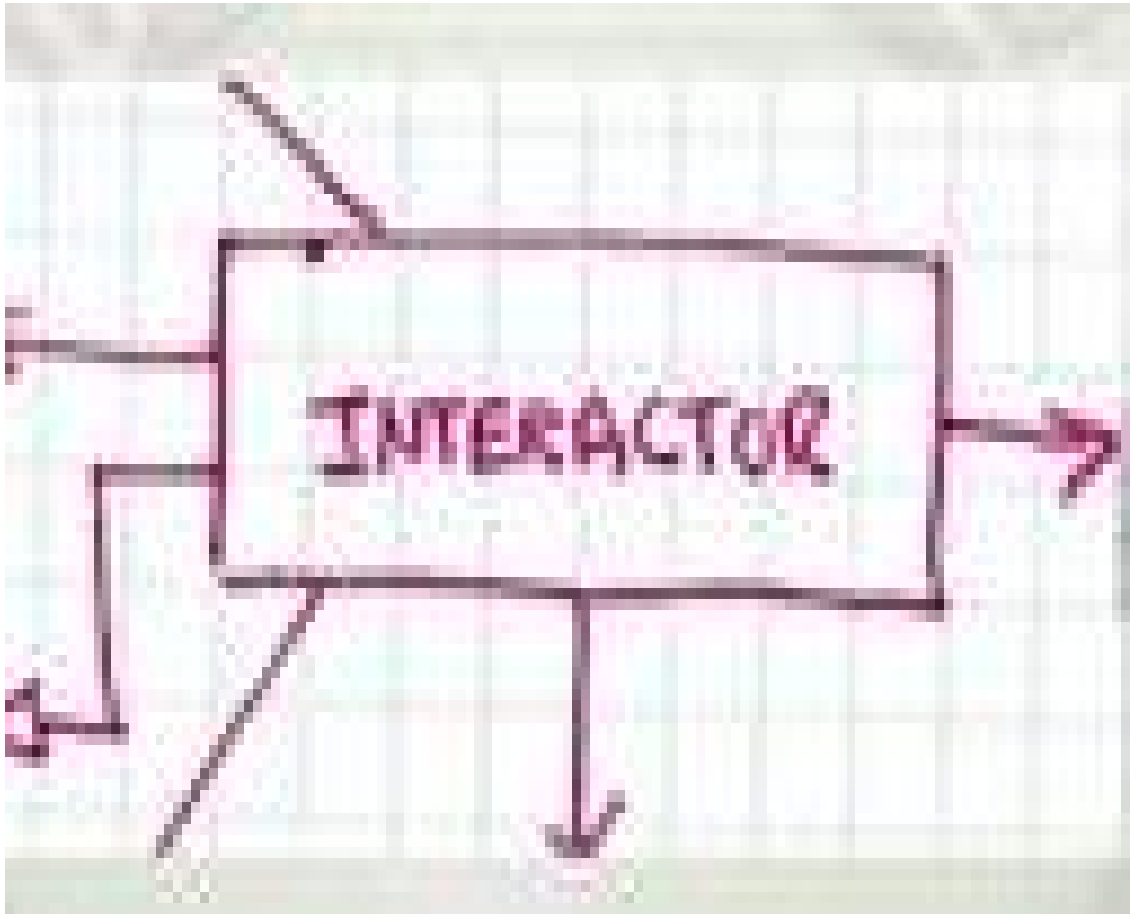
Kui palju me saame äriloogikat arendada ilma, et me teaksime midagi andmebaasist ja kasutajaliidesest?



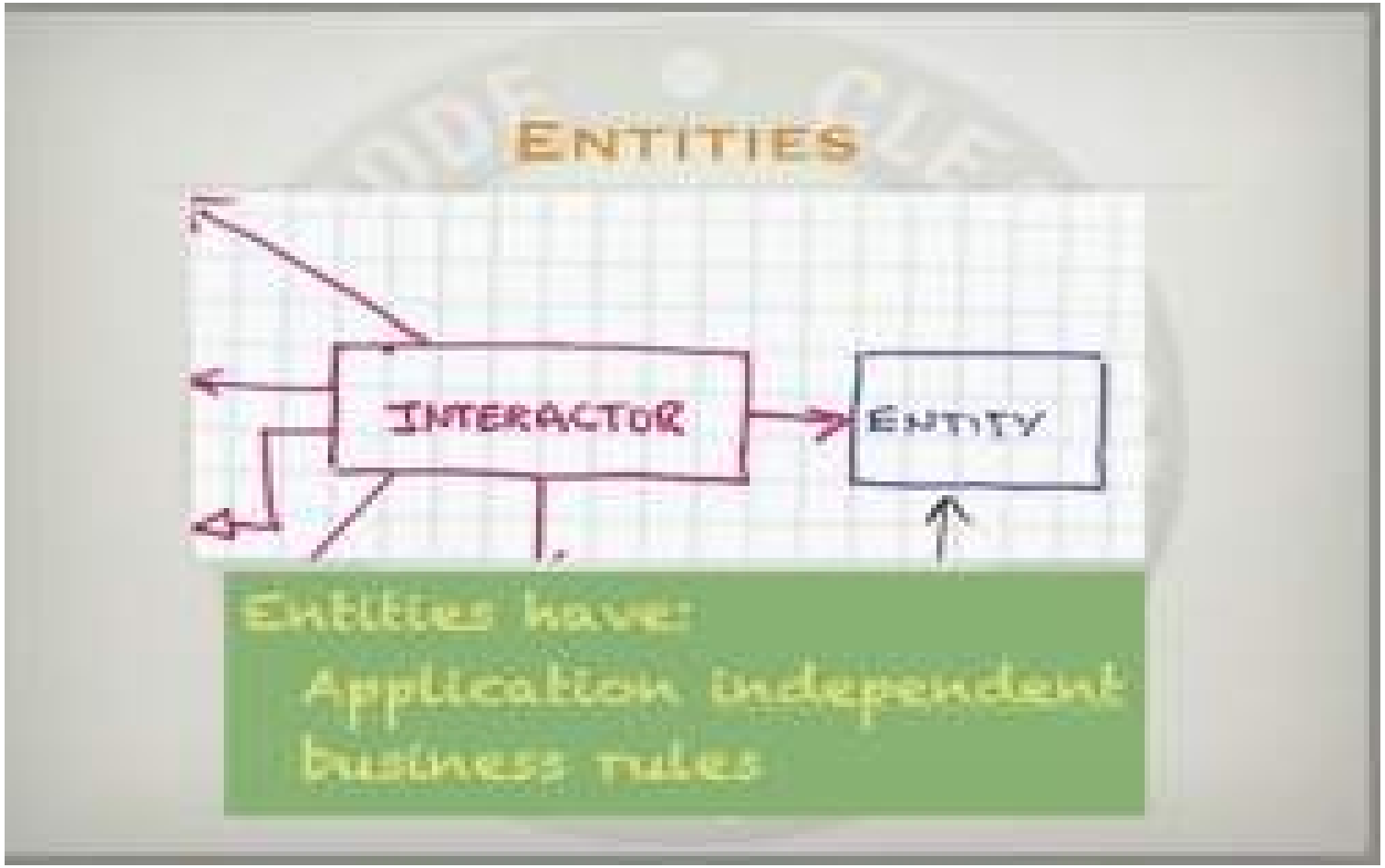
Ja mida tähendab andmebaas ja arvutivõrk ja kasutajaliides?
Sisuliselt sisend/väljund seadmed !



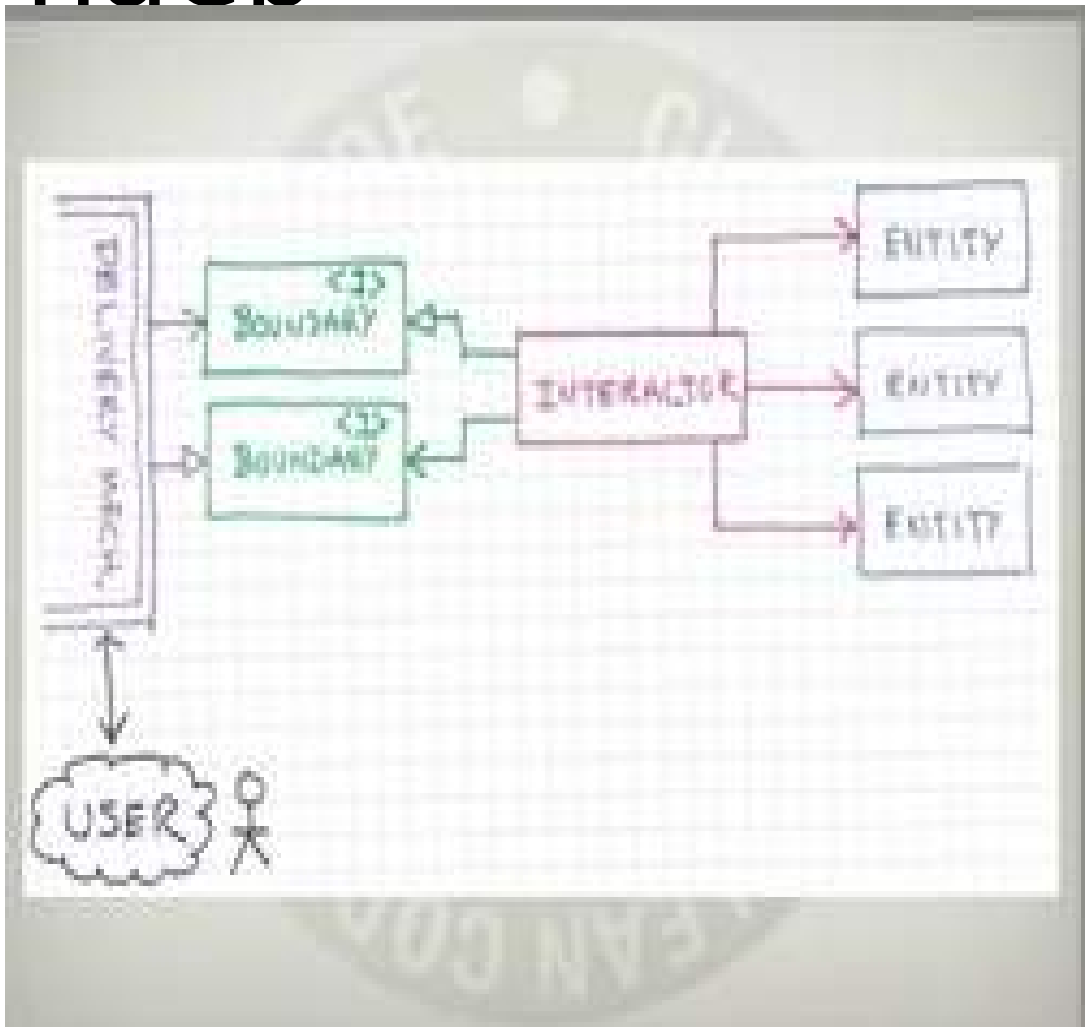
Väga lihtsalt on Use Case (ärinõue) üks klass

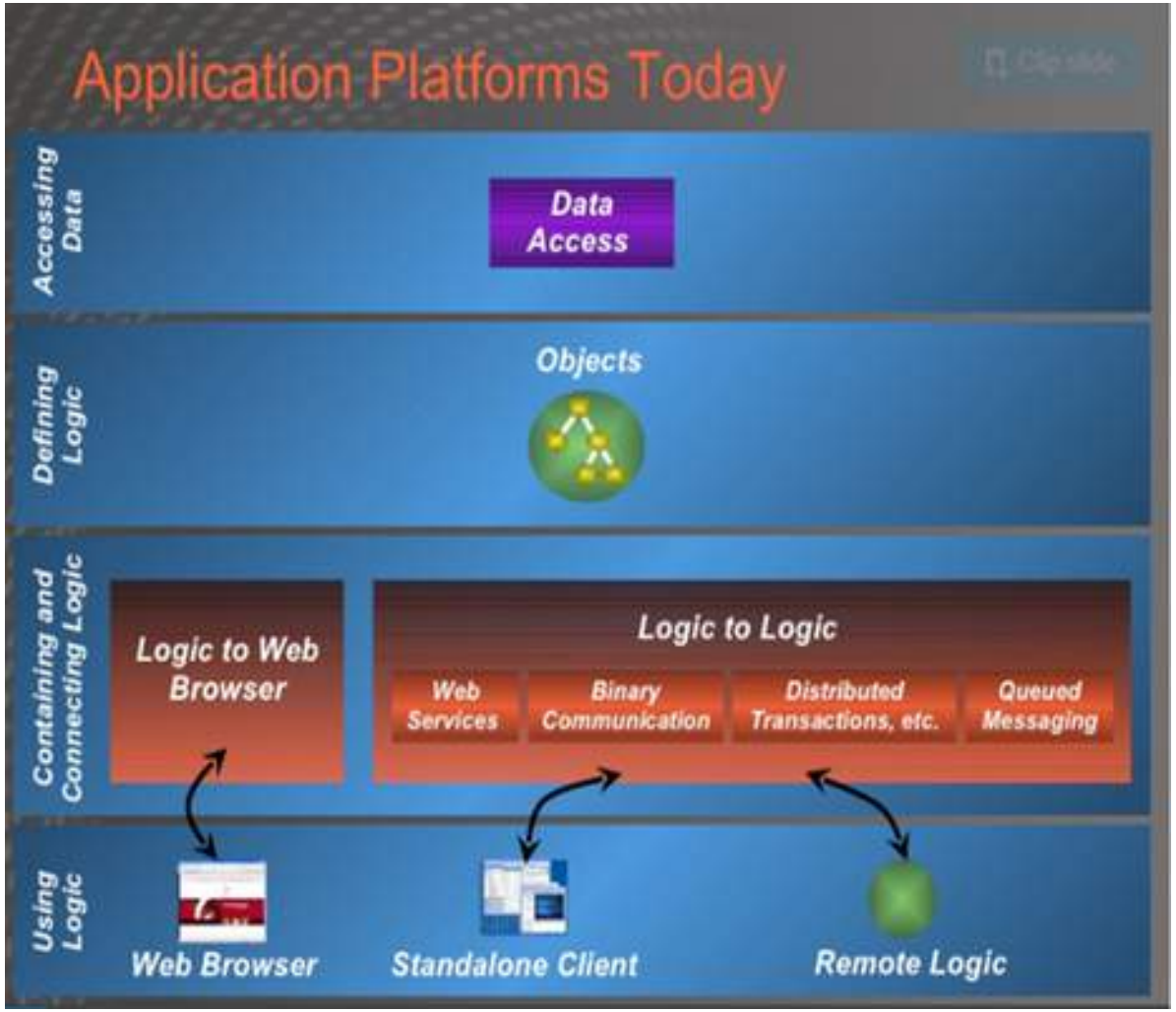


Entity on üks teine klass

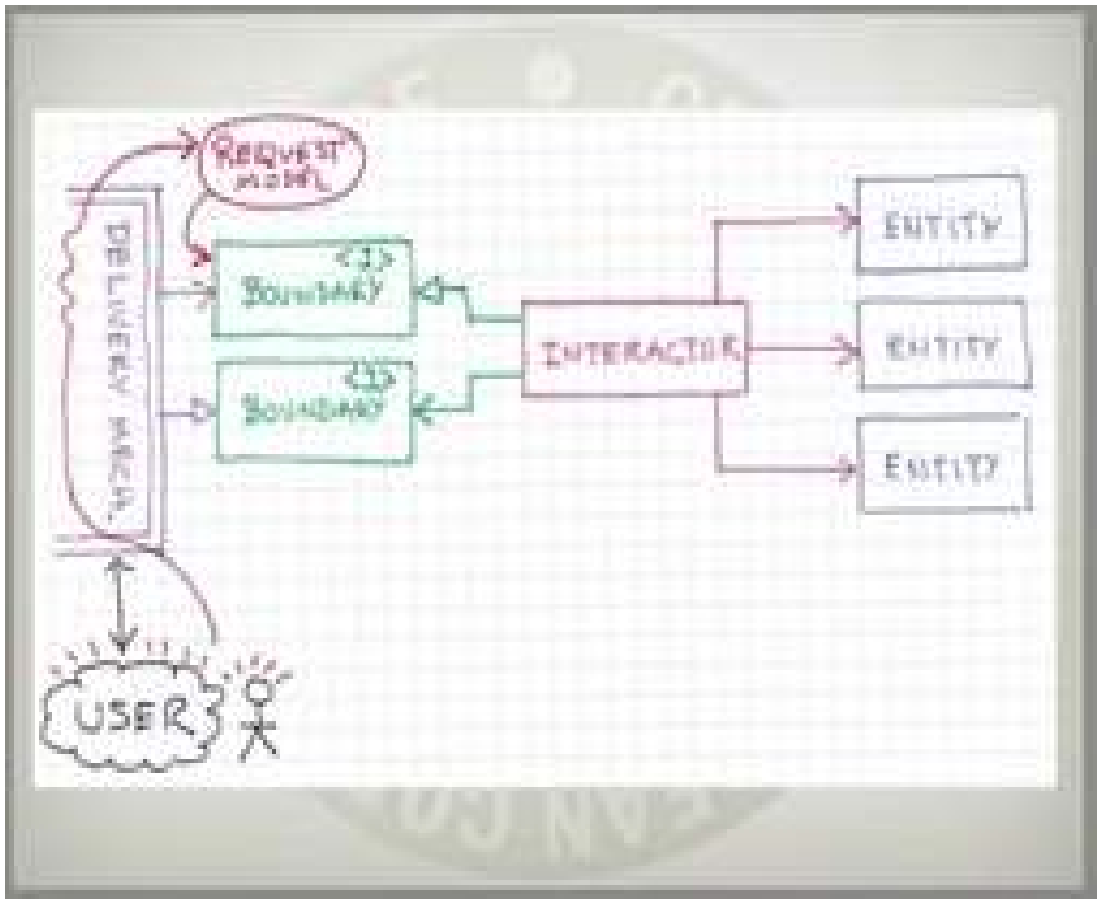


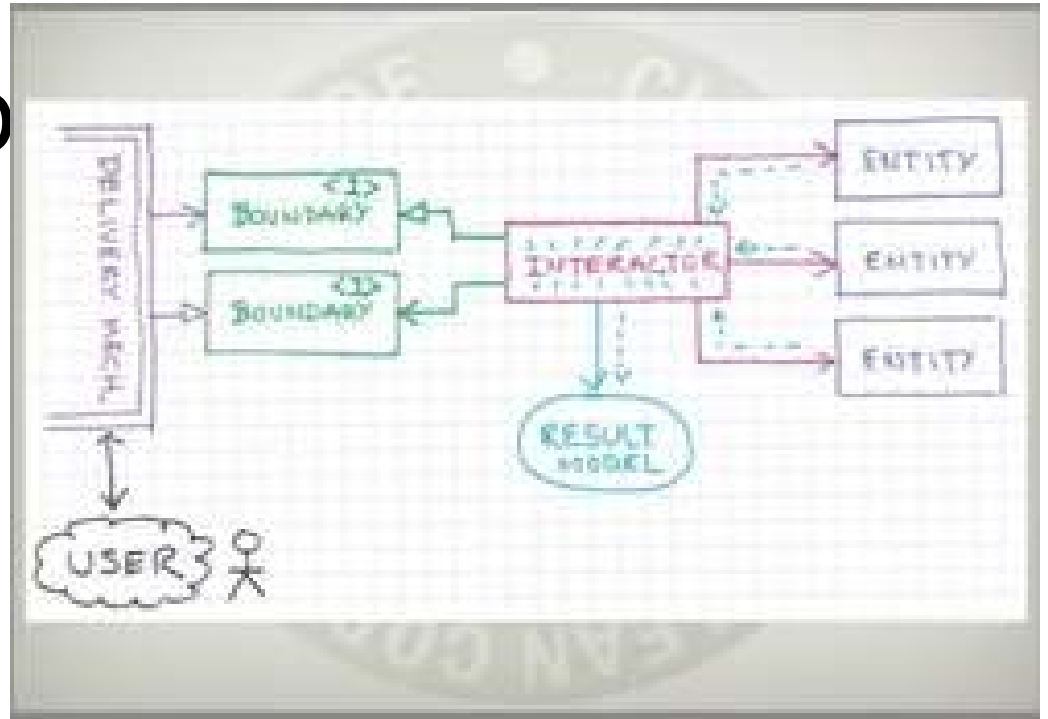
Kuidas siis kogu see süsteem välja näeb

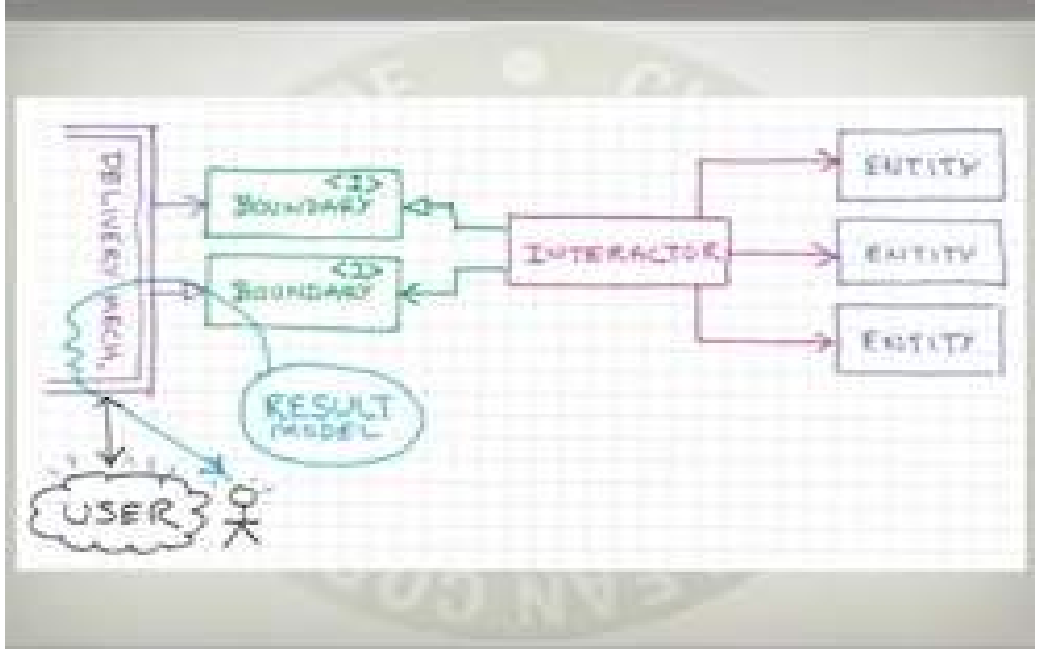




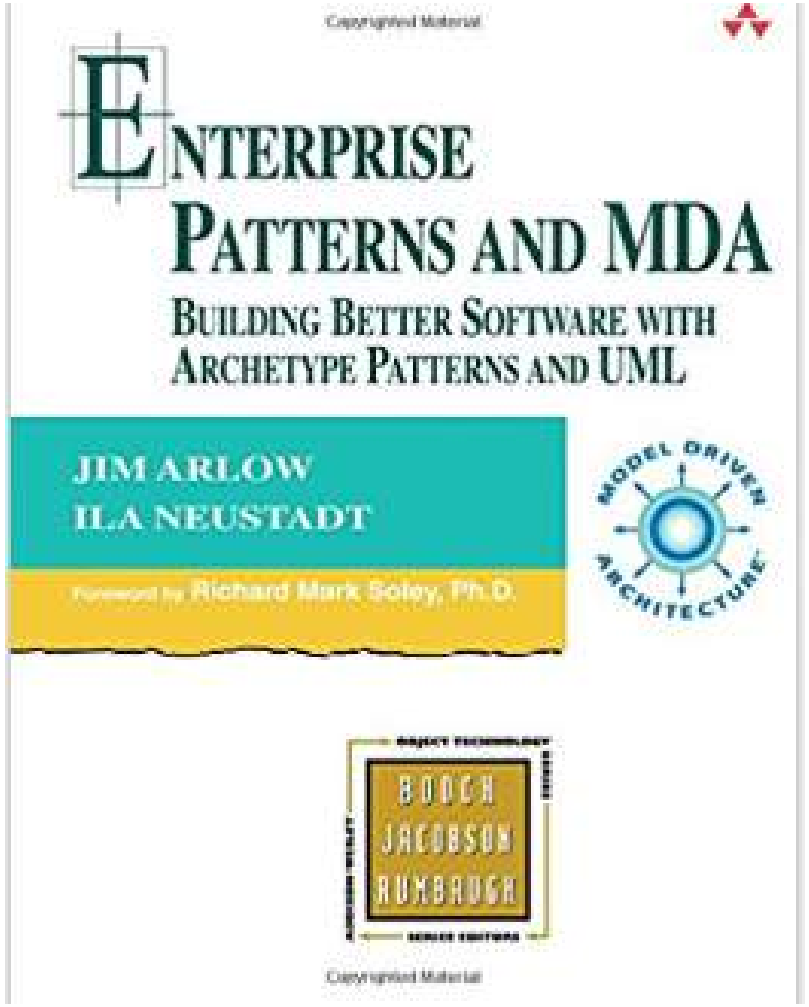
Kuidas siis see asi töötab (Interactor = Use Case) Kas seda saab testida?

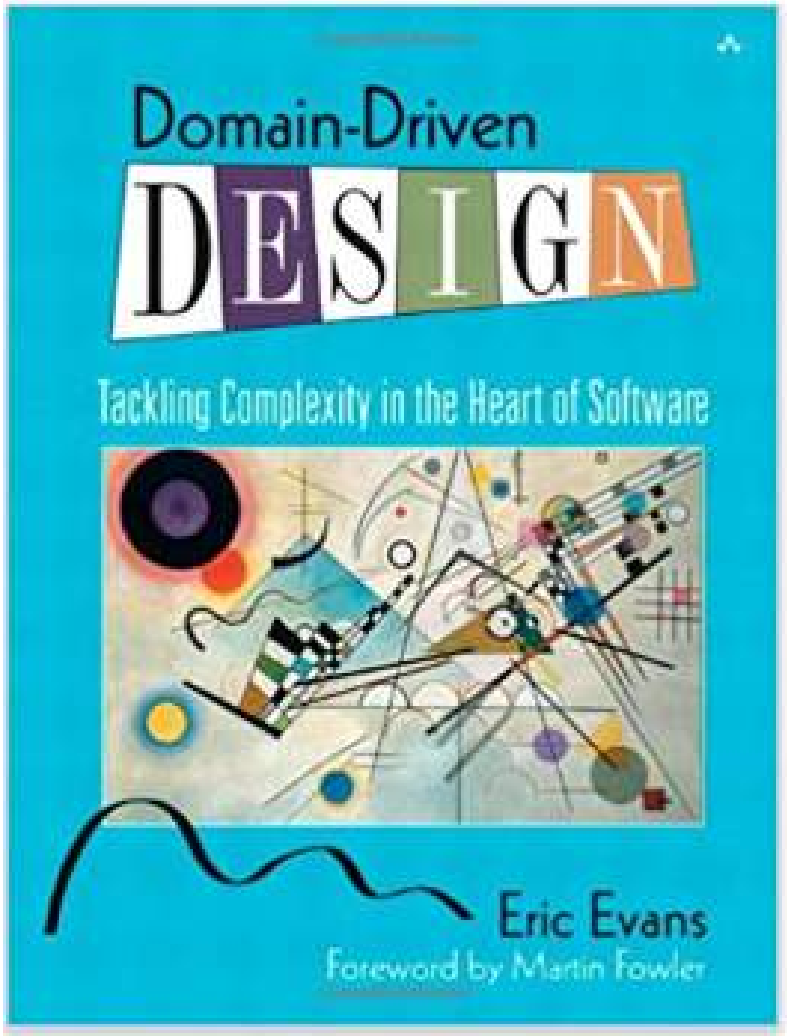




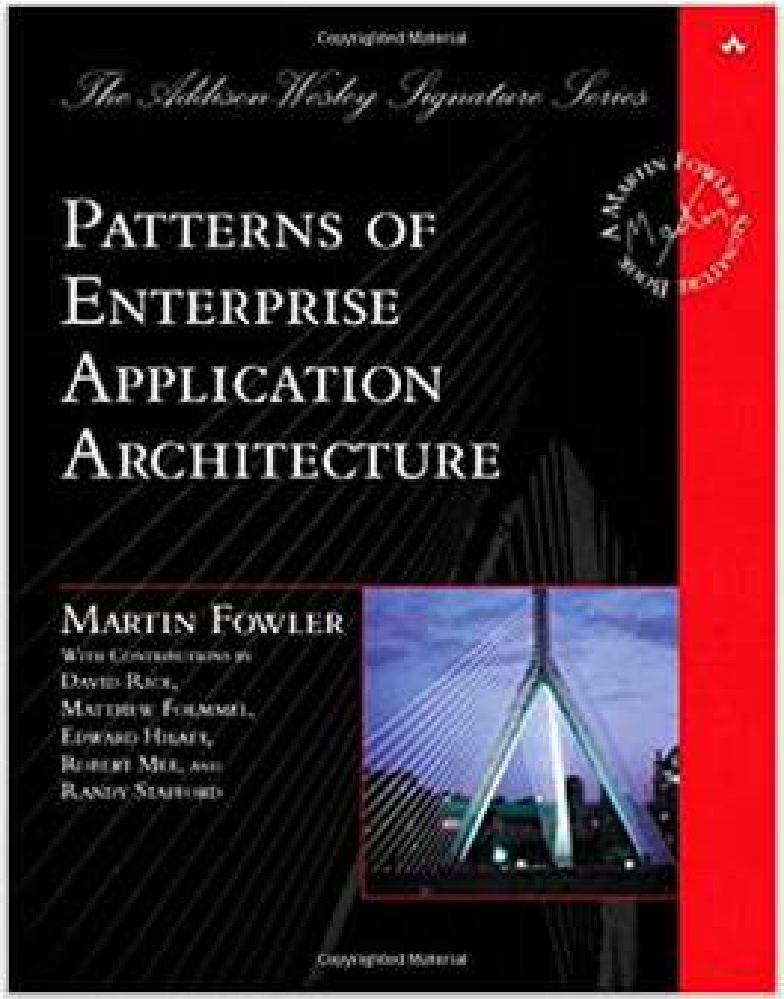


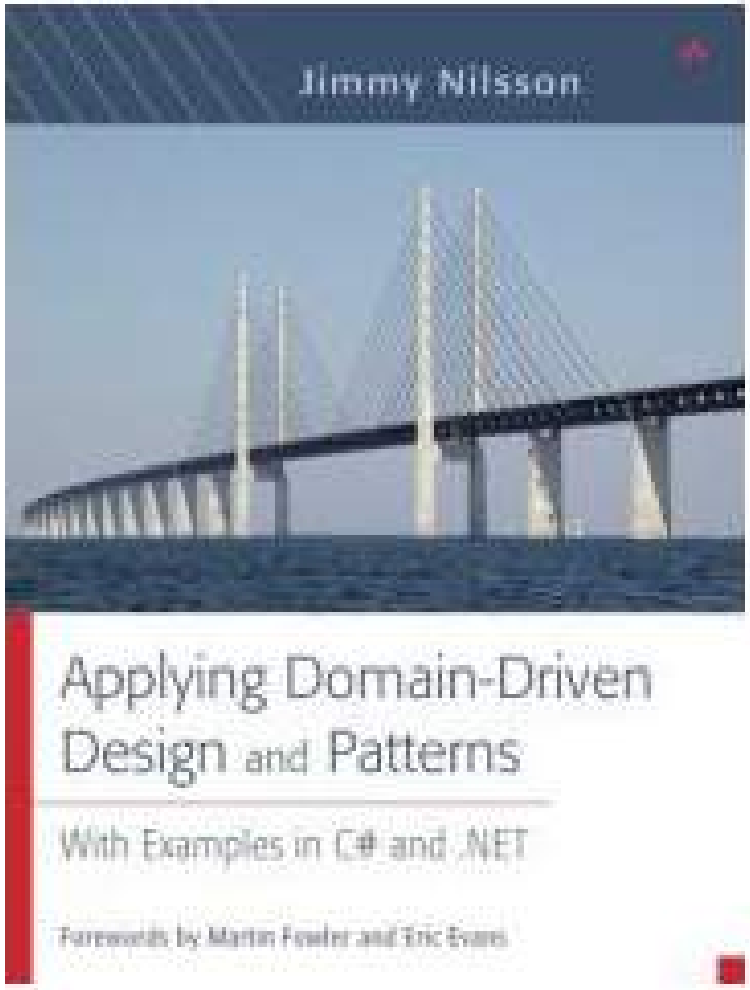
Veel raamatuid





Veel raamatuid





Tänud!